Scheme of work

This scheme of work suggests possible teaching and learning activities for each section of the specification.

There are more activities suggested than it would be possible to teach. Teachers should select activities appropriate to their students and the curriculum time available.

The specification references are shown at the start of each section, whilst the learning outcomes indicate what most students should bep able to achieve after the work is completed. The resources indicate those resources commonly available to schools, and other references that may be helpful. The timings are only suggestions, as are the possible teaching and learning activities, which include references to opportunities for students to explore topics in more depth and breadths it is now written. Resources are only given in brief and risk assessments should be carried out.

General timings

The scheme of work is based on a total of 120 teaching hours. Of these 120 hours, we suggest that:

* 50 hours (approx.) are used for teaching specification sections 3.1 and 3.2 (algorithms and programming)
* 20 hours (approx.) are used for consolidation of programming skills in preparation for writing program code in Paper e
* 40 hours (approx.) are used for teaching specification sections 3.3 to 3.8
* 10 hours (approx.) are reserved for assessments and exam preparation.

Programming and algorithms are at the core of the subject, so we recommend that these topics are taught throughout the course, with the other topics taught alongside them. Based on this scheme of work, in a typical week, approximately two thirds of the time would be spent programming.

**Contents**

You can use the title links to jump directly to the different sections of this scheme of work (Use Ctrl and click to follow the link)

|  |  |
| --- | --- |
| Section title  | Page |
| 3.1 Fundamentals of algorithms and 3.2 Programming | 3 |
| 3.1 and 3.2 Programming consolidation | 19 |
| 3.3 Fundamentals of data representation | 20 |
| 3.4 Computer systems | 28 |
| 3.5 Fundamentals of computer networks | 34 |
| 3.6 Cyber security | 39 |
| 3.7 Relational databases and structured query language | 43 |
| 3.8 Ethical, legal and environmental impacts including privacy | 45 |
| Assessment and exam preparation | 47 |

### 3.1 Fundamentals of algorithms and 3.2 Programming

Total teaching time: 50 hours

Many different approaches can be taken to helping students to understand and develop algorithms and to cultivate the skills required to be able to write computer programs. In the approach outlined, the two topics are taught together. The exercises suggested are grouped together so that students are able to apply the same techniques to solve similar problems, before moving on to learning new techniques.

The problems within each group are graduated in difficulty. At the start of tackling a group of exercises, it would be helpful to give students some program code that they have to modify so that they become familiar with the basic syntax required in their chosen programming language. This could be hand-written or using gap-filling type exercises. However, typing out program code aids familiarity and develops muscle memory so it’s a good idea to have students type out given code (rather than copy and pasting) and then adjust as relevant. Later on, more emphasis should be placed on students designing the solutions themselves. For example, algorithms could be presented in pseudo-code or as flowcharts that students need to implement by writing code in their chosen programming language. Eventually, students should just be set problems where they have to make all the design decisions necessary to produce a fully working program from a problem description.

The emphasis within this course is on solving problems through designing algorithms and, by extension, programming rather than creating fully-working customer-ready programs. All of the required skills can be taught through programming textual user interfaces with no requirement to program applications with graphical user interfaces, although there’s time for this if required.

In the exams, students will need to be able to understand algorithms written in the AQA pseudo-code, so it’s important that students get to see and use this as they learn to program. A guide to the AQA pseudo-code is available on our website. While students are encouraged to write in AQA pseudo-code where requested this is not obligatory. The most important factors are that their pseudo-code is clear and unambiguous. It should also not use structures peculiar to a particular programming language.

Note that for this section of the specification, in the ‘Specification content’ column, only content that’s being covered for the first time or that’s the focus of the exercises is listed. In the later programming exercises, skills introduced earlier will naturally be revisited and reinforced.

Specification reference

3.1.1, 3.2.1, 3.2.2, 3.2.3, 3.2.7

Specification content

* Understand and explain the term algorithm.
* Understand and use string, integer and real data types appropriately.
* Understand how variable declaration and assignment can be used in programs.
* Be able to use addition, subtraction, multiplication and real division.
* Output data and information from a program to a computer display.
* Use meaningful identifier names and know why it’s important to use them.

Learning outcomes

* Apply the listed programming techniques.
* Choose appropriate data types.
* Use meaningful identifier names and know why.
* Understand what an algorithm is and the difference between an algorithm and program.

Suggested timing (lessons)
3 hours

Possible teaching and learning activities

* Introduce students to basic input and output commands, declaring variables (if required by language), and using arithmetic operations.
* Students will also need to be taught basic aspects of the IDE for their programming language[[1]](#footnote-1) eg how to run a program, how to load/save, how error messages are presented and what they mean.
* Introduce students to the idea of an algorithm and that a program is an implementation of an algorithm.
* Exercises could include:
	+ getting the computer to display “Hello World”.
	+ getting the user to type in their name and outputting ‘Hello’ to them (possibly concatenating forename and surname input separately).
	+ doing simple calculations, eg adding three numbers, multiplying two numbers together. This could be done in ‘shell’ mode but will be better contextualised if made into the form of a full program.
	+ doing more complex calculations, eg area of a rectangle, area of a triangle, area of a circle, area of a trapezium. Students could be set the task of completing problems from their maths classes as programs.
	+ students converting between provided code, pseudo-code and flowcharts.

Resource

* [Notes and videos introducing algorithms and example uses](https://www.bbc.co.uk/education/guides/z22wwmn/revision)
* [Notes on variables and data types](https://www.bbc.co.uk/education/guides/zc6s4wx/revision) (and some other concepts not required until later)

Specification reference

3.2.2, 3.2.4, 3.2.5, 3.2.11

Specification content

* Be able to use selection (if, else, else if, case/switch if appropriate)
* Be able to use a range of relational operators ie equal to, not equal to, less than, greater than, less than or equal to and greater than or equal to.it
* Be familiar with and able to use NOT, AND, OR.
* Using nested selection structures.
* Understand what is meant by testing and be able to correct errors in programs and algorithms.
* Be able to select suitable test data that covers normal (typical), boundary and erroneous data. Be able to justify the choice of test data.
* Be able to understand pseudo-code and flowcharts.
* Understand that there are different types of error including syntax and logical errors.
* Be able to identify and categorise errors in algorithms and programs.

Learning outcomes

* Apply the programming techniques listed.
* Choose appropriate test data to use to check their programs.
* Identify, describe and correct both syntax and logical errors.

Suggested timing (lessons)

3 hours

Possible teaching and learning activities

* The focus in this section is on the use of selection statements to determine the path of code execution. Exercises should build in difficulty, starting with simple Yes/No answers using just an If statement then building in complexity in terms of the number of possible outcomes and the complexity of the criteria used.
* Use pseudo-code and flowcharts to illustrate some algorithms which students could then write program code for.
* Whilst completing these exercises, consider choosing test data, which is particularly important in boundary situations of which there are many in these exercises.
* Introduce deliberate errors (a) to introduce students to syntax errors and (b) to demonstrate how logical errors may only be picked up by thorough testing.
* Exercises could include:
	+ exam mark pass/fail.
	+ determining if a person is a child/adult/pensioner based on their age.
	+ allocating an exam grade based on mark ranges.
	+ identifying the biggest of two or three numbers.
	+ identifying if a triangle is scalene, isosceles or equilateral.
	+ classifying the temperature based on a range eg zero0C or below = freezing, above zero0C but 100C or below = warm.
	+ Find the error activities.

Resource

* [Notes and video on use of selection statements](https://www.bbc.co.uk/education/guides/zrxncdm/revision/3)
* [Notes on use of AND, OR and NOT](https://www.bbc.co.uk/education/guides/zc4bb9q/revision/4)
* [Notes on development and testing (including test plans)](https://www.bbc.co.uk/bitesize/guides/z8n3d2p/revision/5)

Specification reference

3.1.1, 3.2.2, 3.2.8, 3.2.9, 3.2.11

Specification content

* Be able to use indefinite iteration with conditions at start and end of loop.
* Be able to use random number generation.
* Be able to use some string handling techniques:
	+ length
	+ position
	+ substring
	+ concatenation
	+ convert character to character code
	+ convert character code to character
	+ string conversion operations.
* Be able to write simple data validation routines.
* Be able to write simple authentication routines.
* Understand and explain the term abstraction.

Learning outcomes

* Apply the programming techniques listed.
* Be able to write simple authentication and validation routines.
* Understand what abstraction is.

Suggested timing (lessons)

4 hours

Possible teaching and learning activities

* Students need to know about indefinite iteration and how to use this in their programming language. For students using Python, which does not have a post-conditioned loop, you should teach how to implement post-conditioned loops as equivalent pre-conditioned loops.
* Students also need to know how to express these types of loop as pseudo-code and flowcharts.
* As students are now starting to tackle more complex problems, the concept of abstraction, ie removing unnecessary details from a problem, could be introduced at this point.
* Exercises could include:
	+ performing simple validation eg that a typed value falls within a range or that an entered value cannot be left blank or is shorter than a minimum length. This can include using the LENGTH string handling technique
	+ adding up a sequence of numbers of unknown length
	+ asking users to enter a password until the correct password is entered, displaying suitable messages
	+ guessing randomly chosen number until they guess correctly, with clues given about whether guess is too high/low
	+ rolling two dice until a double six is scored, counting how many goes this takes
	+ throwing darts and getting a random score on board (game starts at a total and plays with the total decreased by each dart thrown until zero is achieved).

Resource

* [Video on abstraction](https://www.bbc.co.uk/education/guides/zttrcdm/revision)

Specification reference

3.2.2

Specification content

* Be able to use definite iteration.
* Be able to use nested iteration.

Learning outcomes
Be able to use definite iteration.

Suggested timing (lessons)

5 hours

Possible teaching and learning activities

* Intro duce students to the concept of definite iteration and a loop counter. Use pseudo-code and flowcharts to illustrate algorithms.
* Exercises could include:
* counting from one to 10
* displaying a times table, or all times tables
* adding up five numbers (average the same numbers and identify the highest and lowest)
* working out factors of a number using brute-force approach
* identifying prime numbers using brute-force approach.

Resource

* [Notes and videos on use of iteration](https://www.bbc.co.uk/education/guides/zrxncdm/revision/4)

Specification reference
3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.2.6

Specification content

* Understand the concept of data structures.
* Use one-dimensional arrays (or equivalent) in the design of solutions to simple problems.
* Understand that more than one algorithm can be used to solve the same problem.
* Compare the efficiency of algorithms.
* Understand and explain how linear and binary search algorithms work and compare them.
* Understand and explain how bubble and merge sort algorithms work and compare them.
* Use trace tables.

Learning outcomes

* Be able to use one-dimensional arrays.
* Understand the searching and sorting algorithms listed in the specification and be able to compare their efficiency.
* Be able to trace a simple algorithm including up to a single dimensional array to identify output and purpose.
* Understand the searching and sorting algorithms listed in the specification and be able to compare their efficiency.

Suggested timing (lessons)
5 hours

Possible teaching and learning activities

* Introduce students to the concept of a one-dimensional array and give them the opportunity to solve problems using them.
* Cover the four searching and sorting algorithms and give the opportunity to code them, except the merge sort. Before coding these algorithms, it’d be helpful for students to look at them in pseudo-code and to trace their execution in a trace table to ensure that they understand how they function.
* It isn’t expected that concepts such as Big-O or T(n) are introduced. However, counting operations can help demonstrate efficiency.
* Exercises could include:
* inputting a list of names (or other data) and redisplaying them
* inputting a list of parcel weights (total the weights and work out the average, lowest and highest weight)
* searching a dictionary to check whether a word is in it using the linear search method
* improving the dictionary program to use the binary search method
* using the bubble sort algorithm to sort data (eg names) in an array
* looking theoretically at how the merge sort algorithm would perform the same sort (implementing merge sort is beyond GCSE but more able students could attempt this)
* comparing the efficiency of the search and sort algorithms
* representing a game of snakes and ladders using a one-dimensional array to indicate the positions of snakes and ladders.

Resource

* [Notes on data structures and arrays](https://www.bbc.co.uk/education/guides/z4tf9j6/revision/1)
* [Video on searching algorithms](https://www.bbc.co.uk/education/clips/zpbyxsg)
* [Video on sorting algorithms](https://www.bbc.co.uk/education/clips/zt9rnbk)
* [Video of bubble sort using Lego bricks](https://www.youtube.com/watch?v=MtcrEhrt_K0)
* [Video on bubble sort](https://www.youtube.com/watch?v=8Kp-8OGwphY)
* [Simple explanation of Merge Sort video, suitable for GCSE level](https://www.youtube.com/watch?v=EeQ8pwjQxTM)
* [Video comparing linear and binary search](https://www.youtube.com/watch?v=JQhciTuD3E8)

Specification reference
3.1.1, 3.2.2, 3.2.3, 3.2.10

Specification content

* Understand and explain the term decomposition
* Describe the structured approach to programming.
* Explain the advantages of the structured approach.
* Understand the concept of subroutines and be able to use them in programs, including the use of local variables.
* Explain the advantages of using subroutines in programs.
* Integer division, including remainders.

Learning outcomes

* Be able to decompose problems into subroutines and call them and know why this is a good idea.
* Be able to perform integer division, including the use of remainders.

Suggested timing (lessons)

* 4 hours

Possible teaching and learning activities

* Teach students about why, when writing longer programs, it’s useful to decompose them, and the facilities in their programming language to do this. They should also cover the difference between local and global variables. At this stage, parameters and return values can be ignored
* Exercises could include[[2]](#footnote-2):
* making a maths toolkit, with a menu that’s used to call different subroutines to work out (for example) the area of different shapes
* making a program that’ll allow conversion of numbers between different number bases, with different functions being used for different conversions eg binary to decimal
* creating an adventure game with different rooms/actions having different subroutines.
* In all subsequent programs, encourage students to consider how the programs can be decomposed into subroutines.

Resource

* [Video explaining the concept of decomposition](https://www.bbc.co.uk/education/clips/z8nx82p)
* [Notes and video on decomposition, including the use of parameters](https://www.bbc.co.uk/education/guides/z9hykqt/revision) (which isn’t required until later)

Specification reference

3.2.1, 3.2.8, 3.2.10

Specification content

* Use a structured approach to programming, in particular focussing on the use of parameters and return values.
* Use a range of string handling operations from:
* length
* position
* substring
* concatenation
* convert character to character code
* convert character code to character
* string conversion operations.
* Use the char and Boolean data types.

Learning outcomes

* Be able to create and use well-defined interfaces to subroutines, using parameters and return values. Use string handing operations and the Boolean and char data type.

Suggested timing (lessons)

6 hours

Possible teaching and learning activities

* Emphasis should be on passing input to the functions as parameters and using return to pass values back to the calling program. Input/output via the keyboard/screen should not happen within the functions.
* Teach students why this is important, eg in terms of being able to develop and test modules independently and reuse code. Returning tuples (in Python) should be discouraged as it can lead to language-specific pseudo-code.
* Exercises could include:
* developing a function that returns the highest of two numbers and adapting this to find the highest of three numbers or to perform other mathematical operations
* developing a function that indicates whether a number is even or not
* developing a function that works out n factorial (n!)
* developing a function that returns a string that has been encrypted using the Caesar Cipher with a key selected by the user and adding a decryption function (using string position, and conversion between character codes)
* developing a function to convert a string into Morse code
* developing a function that will return a true/false value, indicating if two words are anagrams of each other
* developing a function that, when sent a number, will return a true/false value indicating whether the number is a perfect number or not and using this in a program to search for perfect numbers using brute-force.
* In all subsequent programs, encourage students to consider how the programs can be decomposed into functions with interfaces that use parameters and return values.

Resource
[Notes and audio on structured programming](https://www.bbc.co.uk/education/guides/z9hykqt/revision)

Specification reference
3.22, 3.2.6

Specification content

* Use two-dimensional arrays (or equivalent) in the design of solutions to simple problems.
* Use nested iteration.
* Use of constants.

Learning outcomes

* Use two-dimensional arrays, nested iteration and constants.

Suggested timing (lessons)

10 hours

Possible teaching and learning activities

* Give students the opportunity to write programs using two-dimensional arrays. They’ll need to consider/design how the arrays can be used to represent the problem. Data stored in a two-dimensional array is usually displayed most conveniently using nested loops.
* A range of games can be readily implemented using two-dimensional arrays.
* If students have not yet encountered constants, they could be introduced here, for example, to store the size of a game board.
* Exercises could include:
	+ snakes and ladders
	+ noughts and crosses
	+ battleships.
* Python programmers should use lists rather than importing Array classes for simplicity.

Resource

* [Brief notes on two-dimensional arrays](https://www.bbc.co.uk/education/guides/z4tf9j6/revision/3)

Specification reference

3.2.6, 3.2.1

Specification content

* Use records (or equivalent) in the design of solutions to simple problems.
* Be able to write simple authentication routines

Learning outcomes

* Use records.
* Be able to write simple authentication routines.

Suggested timing (lessons)

6 hours

Possible teaching and learning activities

* Introduce students to the concept of records and why logically grouping related data together is a sensible approach.
* Exercises could include:
* adapt the dictionary program that was written earlier to store equivalent words in two languages in an array of records and perform translation between them
* write an address book program, or a program to keep track of any other data (this data could be saved/loaded from a text file using CSV format)
* adapt the adventure game from earlier to store each room as a record within an array.

Specification reference
3.1.1

Specification content

* Use a systematic approach to problem solving and algorithm creation representing those algorithms using pseudo-code, program code and flowcharts.
* Explain simple algorithms in terms of their inputs, processing and outputs.
* Determine the purpose of simple algorithms.

Learning outcomes

* Students can understand algorithms expressed in pseudo-code and flowcharts and use these methods to write algorithms.
* They can trace the execution of algorithms using a trace table and identify the purpose of an algorithm.
* They can identify the inputs and outputs of an algorithm together with the required processing.

Suggested timing (lessons)

4 hours

Possible teaching and learning activities

* Throughout learning to program, expose students to how algorithms can be expressed using pseudo-code or flowcharts.
* Students need to have some practice at being able to understand and write algorithms using these methods.
* They also need to be able to use trace tables to record the values of variables as an algorithm is stepped through and to be able to identify the purpose of an algorithm by tracing it. This may include use of records, string functions and two-dimensional arrays.
* These skills will be assessed in the exam. It’s useful to teach them in parallel with learning to program (perhaps as homework exercises) but it could also be worth giving students the opportunity to consolidate their ability to apply these skills.
* Students should complete exercises where they have to read and write pseudo-code and flowcharts, complete trace tables and deduce the purpose of algorithms.

### 3.1 and 3.2 Programming consolidation

Total teaching time: 20 hours

Once students have developed their programming skills, it’s important they get the opportunity to consolidate them by working on other programming projects. These could be set by the teacher for the whole class or individually chosen to reflect students’ interests and ability. While there is no longer coursework or non-exam assessment (NEA) programming tasks, by working on larger projects, students are maturing and embedding their programming skills for Paper 1.

When completing consolidation tasks, students need to develop their own skills in analysing problems and designing and testing their solutions, as well as coding them. The more problems that are solved the better – programming the solutions to the problems just allows students to check their solutions work. The specification requires that, for Paper 1, students should have sufficient practice of:

* **structuring** programs into **modular parts** with **clear documented interfaces** to enable them to **design appropriate modular structures for solutions**
* including **authentication** and **data validation** systems/routines within their computer programs
* **writing**, **debugging** and **testing** programs to enable them to develop the skills to articulate how programs work and argue using **logical reasoning** for the correctness of programs in solving specified problems
* **designing** and **applying** **test data** (normal, boundary and erroneous) to the testing of programs so that they are **familiar with these test data types** and the **purpose of testing**
* **refining programs in response to testing outcomes.**

Past and sample coursework assignments set for GCSE coursework are one source of ideas for practising and consolidating programming tasks, as are the programming challenges available on our website.

Able students could also be given the opportunity to extend their skills; for example, if a student learnt how to program in console mode, they could be given the opportunity to develop applications with a graphical user interface. Problems from Paper 1 of the AQA AS/A-level exams may also be used to stretch more able students.

### 3.3 Fundamentals of data representation

Total teaching time: 10 hours

The topics in this section of the specification all require students to applies their skills, so it’s important that they get plenty of opportunities to do this.

Specification reference

3.3.1, 3.3.2

Specification content

* Understand the number bases decimal (base 10) and binary (base 2).
* Understand that computers use binary to represent all data and instructions.
* Understand how binary can be used to represent whole numbers and be able to convert between binary and decimal and vice-versa.

Learning outcomes

* Understand that computers use binary to represent data and instructions.
* Be able to convert between binary and decimal and vice-versa.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities

* Look at how computers store data conceptually as high/low voltage and on and off states and how this can be conceived numerically as binary (may be easier to look at early computers with valves, transistors).
* Review how the decimal system works with 10 digits and place values that are powers of 10 and relate this to how binary works with 2 digits and place values that are powers of 2.
* Show how a binary number can be converted to decimal by adding the place values of columns with 1s in.
* Show how decimal can be converted to binary by working from left to right.
* Consider the highest and lowest decimal value that can be stored in 8 bits.
* This is a topic that students must practise, so they need to complete conversion exercises, possibly some in class and some for homework.

Resource

* [Notes on binary and conversion](https://www.bbc.co.uk/bitesize/guides/zwsbwmn/revision/1)
* [Binary conversions game](https://learningnetwork.cisco.com/docs/DOC-1803)

**Specification reference**
3.3.1, 3.3.2

Specification content

* Understand the number base hexadecimal (base 16).
* Understand how hexadecimal can be used to represent whole numbers and be able to convert between decimal and hexadecimal as well as binary and hexadecimal.
* Understand why hexadecimal is often used in computer science.

Learning outcomes

* Be able to convert between decimal and hexadecimal as well as binary and hexadecimal.
* Understand why hexadecimal is often used in computer science and give examples of where it is used.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities

* Consider why binary is not easy for humans to use (eg long strings of digits, easy to transpose, hard to remember).
* Explain why hexadecimal is a good shorthand for binary (4 bits = 1 hex digit) and look at where hex is used eg colour codes, MAC addresses, memory editors.
* Look at methods for converting between decimal and hexadecimal and vice-versa (remember only 8-bit numbers are needed).
* Look at the quick method for converting between binary and hexadecimal and vice-versa in groups of 4 bits.
* Students need to complete plenty of example conversion exercises in class and for homework.
* It’s worth considering whether your students would learn a direct decimal🡪hexadecimal🡪decimal conversion method or would be better using binary as an intermediary.

Resource

* [Online notes on conversions](https://www.bbc.co.uk/education/guides/zp73wmn/revision)

Specification reference
3.3.3

Specification content

* Know the units that are used to measure quantities of bytes.

Learning outcomes

* Students know the units bit, byte, kilobyte, megabyte, gigabyte and terabyte.
* Students know the names, symbols and corresponding values for decimal prefixes and are able to compare quantities.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities

* Explain the names of the measurements used for quantities. Consider a comparison with measurements for distance where different but related measurements are used depending on the magnitude of the distance being measured (eg cm, m, km).
* Emphasise that this specification uses the SI definitions of the units which are powers of 10, but refer to the historical definitions using powers of 2, which students may be familiar with.
* Look at measurements of sizes of typical things eg RAM in a computer, size of a hard disk, download allowances.
* You could set students some exercises working out file sizes or converting between units.
* Give students exercises for comparing between prefixes – eg what is larger, 3,000 Megabytes or 4 Gigabytes?

Specification reference
3.3.4

Specification content

* Be able to add together up to three binary numbers.
* Be able to perform logical shifts.
* Describe situations where binary shifts can be used.

Learning outcomes

* Be able to add together three binary numbers.
* Be able to perform logical shifts.

Suggested timing (lessons)

2 hours

Possible teaching and learning activities

* Show students the method for completing binary addition of three numbers, including how to deal with multiple carries.
* Then they should complete some exercises to practise this.
* Then show students how a binary shift can be used to double/ approximately halve a number.

Resource

* [Notes on binary addition](https://www.bbc.co.uk/education/guides/zjfgjxs/revision) (note that negative numbers are not required)
* [Binary addition video](https://www.youtube.com/watch?v=ypqYoFbPfTk)

Specification reference

3.3.5

Specification content

* Understand character sets including ASCII and Unicode and the advantages of Unicode.
* Understand that character codes are commonly grouped and run in sequence within encoding tables.

Learning outcomes

* Understand character sets including ASCII and Unicode and the advantages of Unicode.
* Be able to use a character table to convert a message from binary to a character set and vice-versa.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities

* Look at the ASCII table.
* Complete exercise converting a message from binary to characters and vice-versa.
* Note how similar characters are in blocks eg all capital letters.
* Consider limitations of ASCII (limited number of characters) and look at how Unicode solves these.
* This topic could be linked to programming through the use of the programming language commands for conversion between character codes and characters.

Resource

* [Online notes on character sets](https://www.bbc.co.uk/education/guides/zp73wmn/revision/5)
* [The ASCII table](http://www.asciitable.com/)
* [Official Unicode website](http://www.unicode.org/)
* [Unicode character tables](http://www.unicode.org/charts/)

Specification reference

3.3.6

Specification content

* Understand how images can be represented as bitmaps, including key terms.
* Be able to calculate file sizes.
* Be able to convert between binary and image data for simple images.

Learning outcomes

* Define key terms eg pixel, colour depth.
* Calculate file sizes for bitmap images.
* Convert between an image and binary and vice-versa.

Suggested timing (lessons)

1.5 hours

Possible teaching and learning activities

* Look at bitmap images using a graphics package, use zoom to identify pixels and colours (possible link to hex).
* Introduce colour depth by considering how different patterns of 0s and 1s could be used to represent colours. A colour depth of n bits allows 2n colours.
* Perform some exercises where students have to convert small images between images and binary data and vice-versa.
* Explain how to calculate the size of an image file and then students complete some sample calculations.

Resource

* A bitmap image editor eg Paint.
* [Online notes and test](https://www.bbc.co.uk/education/guides/zqyrq6f/revision) (note: vector graphics are not required)

Specification reference
3.3.7

Specification content

* Understand analogue sound must be sampled and converted to digital form for storage and processing.
* Describe how sound is represented using sample rate and sample resolution.
* Calculate sound file sizes.

**Learning outcomes**

* Understand the difference between analogue and digital.
* Be able to recognise analogue and digital quantities.
* Be able to explain how sample rate and sample resolution affect sample quality and file size.
* Be able to calculate file sizes for sound files.

Suggested timing (lessons)

1.5 hours

Possible teaching and learning activities

* Discuss difference between analogue and digital quantities.
* Look at how sound can be represented electronically as a waveform – a package such as Audacity can be used to allow students to look at sounds and record their own.
* Use a graph to show how the sampling process works and how sample quality and size would be affected by changing sample rate and sample resolution.
* Perform calculations of sound file sizes.
* Students should carry out exercises that involve identifying analogue and digital quantities, converting between an analogue waveform and digital samples and calculating sound file sizes.

Resource

* A sound editing package such as [Audacity](http://www.audacityteam.org/)
* [Video](https://www.youtube.com/watch?v=HqHIOA-Fcuw) (goes beyond GCSE level)

Specification reference

3.3.8

Specification content

* Explain what data compression is, why it’s used and why there are different methods used.
* Be able to explain how Huffman coding works and know how to use a tree to decompress data using Huffman coding and calculate how many bits are used vs how many bits are stored using uncompressed ASCII data.
* Be able to compress/ decompress data using RLE.

Learning outcomes

* Explain what data compression is, and why it’s used.
* Be able to decompress data using Huffman coding and calculate how many bits are used compared to using uncompressed ASCII.
* Be able to compress/decompress data using RLE.

Suggested timing (lessons)

1.5 hours

Possible teaching and learning activities

* Students could try creating ZIP files or comparing the size of JPEG (compressed) and Bitmap files of the same image to see the effect of compression.
* In discussion, consider why compression is useful – either in the context of transmission or storage of data eg faster downloads, more photos on memory cards etc.
* RLE is the simplest of the two techniques so it’s best to cover this first. Look at how it can be used with small images and get students to try compressing small bitmaps using it. Consider why it isn’t suitable for some images and many types of data.
* Look at Huffman coding and the concept of variable-length codes with more common characters having shorter codes.
* Students should try using a Huffman tree to decode some text stored as binary data. At this point, a calculation of how much memory was saved compared to using 7-bit ASCII can be made.
* Students don’t need to be able to build a Huffman coding tree although doing so can aid understanding.
* Students need to complete practice exercises compressing and decompressing data using both RLE and Huffman coding and calculating how much memory is saved when Huffman coding is used.
* There are lots of videos available illustrating these techniques.

Resource

* [Compression methods and RLE video](https://www.youtube.com/watch?v=pbhe7DXwcIQ)
* [RLE video](https://www.youtube.com/watch?v=ypdNscvym_E)
* [Huffman coding video](https://www.youtube.com/watch?v=apcCVfXfcqE)

#### 3.4 Computer systems

Total teaching time: 9 hours

Specification reference
3.4.1, 3.4.3

Specification content

* Define the terms hardware and software and understand the relationship between them.
* Explain what’s meant by system software and application software and be able to give examples of them.
* Understand the need for and functions of the OS and utility programs.

Learning outcomes

* Define the terms hardware and software and understand the relationship between them.
* Explain what’s meant by systems software and application software and be able to give examples of them.
* Understand the need for and functions of the OS and utility programs.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities

* This is very much a theory topic so is probably best delivered by the teacher talking and discussing with the class.
* For the first point, students simply need to know that hardware is the electronic or electro-mechanical components of the computer and that software are the programs that run on the hardware and tell it what to do to perform a task.
* Students need to know that application software completes user-oriented tasks that the user would need to do with or without a computer whereas system software performs tasks related to the management of the computer system.
* Students need to know/understand that the OS manages processor(s), memory, I/O devices, applications and security but don’t need to know how.
* A utility is a program that helps manage a computer but isn’t core to its operation eg a compression program, a virus-checker. It might be useful to make students aware that utilities are increasingly being bundled with the OS.
* More able students might enjoy researching machines such as the PDP/8 which didn’t have an OS and consider how such machines worked.
* Practice tasks arranging software into categories can be useful.

Resource

[Operating systems online notes](https://www.bbc.co.uk/education/guides/ztcdtfr/revision/1)

Specification reference
3.4.2

Specification content

* Construct truth tables for NOT, AND, OR, XOR gates,
* Construct truth tables for simple logic circuits and interpret them.
* Create, modify and interpret simple logic circuit diagrams.
* Create and interpret simple Boolean expressions made up of NOT, AND, OR and XOR operations.
* Convert between Boolean expression and logic circuits.

Learning outcomes

* Be able to construct truth tables for gates and circuits.
* Be able to draw logic circuits to represent a simple logic problem.
* Be able to create simple Boolean expressions and convert between Boolean expressions and logic circuits.

Suggested timing (lessons)

3 hours

Possible teaching and learning activities

* Consider the basic operations of AND, OR, XOR and NOT (students may have already come across these in the context of programming or databases depending on the order in which the sections are taught).
* Look at truth tables for each gate.
* Draw a logic circuit and then build a truth table for it.
* Students should then try some exercises completing truth tables for different logic circuits.
* Introduce the idea of drawing a logic circuit to represent a specific problem. Students should then try to draw logic circuits for a few problems. This could be done on paper, using an online logic circuit simulator or physically using electronics or tools such as logic goats.
* It isn’t required for the specification but it’d be useful to link this in to hardware and the design of the processor by explaining how gates can be combined to make a processor or memory. XOR can be demonstrated using a mask to change between upper- and lower-case ASCII.

Resource

* [Online logic gate simulator](https://academo.org/demos/logic-gate-simulator/)
* [Notes and video on logic circuits and Boolean expressions](https://www.bbc.co.uk/education/guides/zc4bb9q/revision)

Specification reference
3.4.4

Specification content

* Know and explain the differences between low-level and high-level languages.
* Explain the differences between low-level languages (assembly language and machine code).
* Understand the need to translate high-level or assembly languages.
* Understand that machine code is expressed in binary and is specific to a processor or family of processors.
* Understand the advantages/ disadvantages of low-level vs high-level language programming.
* Understand and explain the differences between, and times to use, interpreters, compilers and assemblers.

Learning outcomes

* Be able to describe and discuss low and high-level languages, their advantages and their uses.
* Be able describe and discuss interpreters, compilers and assemblers and when they could be used.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities

* This is a largely theoretical lesson which is most likely to be teacher-led.
* Students could research the subject matter and produce comparison charts.
* Mini case studies could be used to identify the most appropriate solutions and to engender discussion.

Specification reference
3.4.5

Specification content

* Explain role of main memory, components of CPU, within the Von Neumann architecture:
	+ arithmetic logic unit
	+ control unit
	+ clock
	+ register
	+ bus.
* Explain the effect of clock speed, number of cores and cache size on performance of the CPU.
* Understand and explain the fetch-execute cycle.

Learning outcomes

* Explain role of main memory, components of CPU, buses and the impact of certain changes.
* Understand and explain the fetch-execute cycle.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities

* A good way to introduce this is to have old PCs that students can look inside of to identify the component parts. This could be done with photographs but having real PCs makes it more interesting.
* The role of the components needs to be explained.
* Students only need a high-level understanding of the fetch-execute cycle. They don’t need to know the details of register operations etc.
* A range of online simulators can be used to illustrate this.

Resource

* [Online notes, including some videos](https://www.bbc.co.uk/education/guides/zmb9mp3/revision) (note some content not required)
* [Notes on factors affecting processor performance](https://www.bbc.co.uk/education/guides/zws8d2p/revision/2)
* [Animation of fetch-execute cycle](https://www.youtube.com/watch?v=IL44-Mfp8x4) (more detail than needed)

**Specification reference**
3.4.5

Specification content

* Understand difference between main memory and secondary storage and between RAM, ROM, cache and a register, what they’re used for and why they’re required.
* Be aware of why secondary storage is needed and the different types of secondary storage.
* Explain the operation of solid state, optical and magnetic storage.
* Discuss their relative advantages.
* Explain what cloud storage is and compare it to local storage.

Learning outcomes

* Understand difference between main memory and secondary storage and between RAM and ROM. Be able to explain volatile and non-volatile.
* Explain the operation of solid state, optical and magnetic storage.
* Discuss their relative advantages.
* Explain what cloud storage is and compare it to local storage.

Suggested timing (lessons)

2 hours

Possible teaching and learning activities

* This isn’t a very practical topic. Most of the content is probably best explained to the students by the teacher, although students could be asked to research parts of it eg what cache is and how it improves performance.
* With regard to RAM and ROM, it’s helpful to focus on their uses.
* It’s useful to have physical devices for students to look at here – a disassembled hard disk drive and CD-ROM drive or similar. There’s less of interest that can be seen inside a solid state drive.
* There are also lots of animations available on the Internet on websites such as [howstuffworks.com](http://www.howstuffworks.com/), which illustrate the principles behind the operation of these devices.
* Students could make a presentation to explain how each device works.
* The relative advantages of the devices should be considered in relation to criteria such as maximum capacity, cost per megabyte, robustness, power consumption and portability.
* Many students will be familiar with using cloud storage such as OneDrive or Apple or Google’s cloud storage systems, so this aspect of the specification would work well as a discussion with students explaining what they use it for and considering the practical benefits they’ve seen themselves but also the risks.

Resource

* [Online notes, including some videos](https://www.bbc.co.uk/education/guides/zmb9mp3/revision) (note some content not required)
* Disassembled storage devices, websites such as:
	+ [How hard disks work](https://computer.howstuffworks.com/hard-disk.htm)
	+ [How solid-state drives work](https://computer.howstuffworks.com/solid-state-drive.htm?srch_tag=lezzth5sgczivnehami6eaehyutl2ju4)
	+ [How CDs work](https://electronics.howstuffworks.com/cd.htm)
* [Storage device summary](https://www.bbc.co.uk/education/guides/zxgkxnb/revision/6)
* [The cloud](https://computer.howstuffworks.com/cloud-hard-disk.htm)

Specification reference
3.4.4

Specification content
Understand the term ‘embedded system’ and explain how an embedded system differs from a non-embedded system.

Learning outcomes

* Explain what an embedded system is and how an embedded system differs from a non-embedded system.
* Give examples of embedded systems.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities

* This is a relatively small topic. Students need to understand that many computer systems are embedded in other devices and the constraints and differences that this produces when compared with non-embedded systems.
* Give students some scenarios (eg washing machine) and ask them to consider what functionality the system would need and why a non-embedded system wouldn’t be suitable.
* Differences such as processor speed, amount and type of main memory, secondary storage, input and output devices and upgradeability could be considered.
* More able students could investigate the part embedded systems play within the Internet of Things.

#### 3.5 Fundamentals of computer networks

Total teaching time: 6 hours

Many of the topics in this section are quite theoretical, so could be delivered through discussion and students using textbook/notes. It’s important that students have the opportunity to demonstrate their understanding by answering questions. Some practical networking could be done using, for example, Raspberry Pi computers, which students can build a network from themselves, but this is not a requirement.

Specification reference
3.5

Specification content

* Define what a computer network is.
* Discuss the benefits and risks of computer networks.
* Understand that networks can be wired or wireless.
* Discuss the benefits and risks of wireless networks as opposed to wired networks.

Learning outcomes
Students should be able to explain what a computer network is, discuss risks and benefits of networks and the relative merits of wired and wireless networking.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities

* Students will have direct experience of using networks, both wired and wireless, so this makes a good discussion topic: pros and cons of having a network and also of wired vs wireless networks.
* Devices such as Raspberry Pis could be used to build a network if it’s desired that students have some practical experience.

Resource
[Online notes and test](https://www.bbc.co.uk/education/guides/zh4whyc/revision) (note also covers some topics that aren’t required)

Specification reference
3.5

Specification content

* Describe the LAN, WAN and PAN types of computer network.
* Explain the star and bus physical network topologies.

Learning outcomes
Students can describe LAN, WAN and PAN and understand star and bus topologies, including their relative merits.

Suggested timing (lessons)

2 hours

Possible teaching and learning activities

* Differences between LAN and WAN should be considered in terms of size, ownership and the hardware used.
* Topologies are best visualised; it’s worth noting that physical bus networks have limited applications nowadays.
* This topic can be taught as a discussion or there are many online videos and resources.
* Students can build a bus and a star network out of string and coat hangers and demonstrate problems by cutting connections.

Resource

* [Online notes and test](https://www.bbc.co.uk/education/guides/zh4whyc/revision) (note also covers some topics that are not required).
* [Video](https://www.youtube.com/watch?v=WwEZR2vU1UA) (covers more topologies than needed).

Specification reference
3.5

Specification content

* Define the term ‘network protocol’.
* Explain the purpose and use of common network protocols including: Ethernet, Wi-Fi, TCP, UDP, IP, HTTP, HTTPS, FTP, SMTP, IMAP.

Learning outcomes

* Students understand and can describe what a protocol is.
* Students can explain the purpose of the protocols and their use. They don’t need to know any technical details about implementation.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities

* This topic is very theoretical and is probably best taught with students reading notes or the teacher delivering a presentation. Students should then answer questions that test their understanding. It’s possible to demonstrate the use of some of the protocols, for example by using Telnet to open connections to a web server or email server, but this isn’t required for GCSE.
* Students could carry out short research tasks to identify core reasons of the protocols.
* Some online resources are also available.

Resource
[Online notes](https://www.bbc.co.uk/education/guides/zh4whyc/revision) (pages 5 and 6 have basic coverage of protocols).

Specification reference
3.5

Specification content

* Understand the need for, and importance of, network security.
* Explain the following methods of network security: authentication, encryption, firewall, MAC address filtering.

Learning outcomes
Students understand why security is important on networks (more so than standalone computers) and the listed security measures.

Suggested timing (lessons)
1 hour

Possible teaching and learning activities

* This topic can be taught theoretically or, if the teacher has access to this, students could be shown how some of these measures are used in school, eg firewall rules used.
* Roleplay can also be a useful tool with some students playing data packets and some students playing the security tool.

Resource

* [Online notes and test](https://www.bbc.co.uk/education/guides/zs87sbk/revision)
* [Video showing use of MAC address access list](https://www.youtube.com/watch?v=SMf-2LjikPA)s
* [Very short video on firewalls](https://www.youtube.com/watch?v=wf2ikTtz_gk)

Specification reference
3.5

Specification content

* Describe the 4 layer TCP/IP model.
* Understand that the HTTP, HTTPS, SMTP, IMAP and FTP protocols operate at the application layer.
* Understand that the TCP and UDP protocols operate at the transport layer.
* Understand that the IP protocol operates at the network layer.

Learning outcomes
Students should know what the four layers are and some functions of each layer, together with which of the protocols listed operate at which layer.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities

* This topic is fairly theoretical. Students could use textbooks, online notes or videos to learn from.
* They need to understand why a stack is used (abstraction), what the four layers are and some functions of each layer of the stack and at which layers the listed protocols work.

Resource
[Video](https://www.youtube.com/watch?v=mRd79VFkSvQ) on TCP/IP

#### 3.6 Cyber security

Total time for teaching this section: 4 hours

Specification reference
3.6.1, 3.6.2

Specification content

* Define the term cyber security and be able to describe the main purposes of cyber security.
* Understand and be able to explain the following cyber security threats:
	+ social engineering techniques
	+ malicious code
	+ weak and default passwords
	+ pharming
	+ misconfigured access rights
	+ removable media
	+ unpatched and/or outdated software.
* Explain what penetration testing is and what it is used for.

Learning outcomes

* Be able to explain cyber security and the cyber security threats covered by the specification.
* Be able to explain penetration testing and what it is used for.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities

* This topic works well as a class discussion as most students will be familiar with some of these topics from their own personal experiences.
* Students could make a presentation, each focusing on one or more topics.

Resource

* [Documentary on cybercrime in the UK](https://www.youtube.com/watch?v=8NAbV6w-KUQ)
* [Five of the worst computer viruses](https://www.youtube.com/watch?v=DF8Ka8Jh0BQ)
* [Notes on some topics of computer security](https://www.bbc.co.uk/education/guides/zs87sbk/revision)
* [Video on penetration testing](https://www.youtube.com/watch?v=q2t91jLmh3k)

Specification reference
3.6.2.1

Specification content

* Describe what social engineering is.
* Explain the following forms of social engineering: blagging, phishing, shouldering
* Describe how social engineering can be protected against.

Learning outcomes
Be able to describe social engineering and methods that are suitable for protecting from social engineering threats.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities
This is an excellent opportunity for students to discuss personal experiences. Teachers may have examples of phishing attempts. The lesson could be started as simply as asking every student to write down their password and give it to their friend. The responses of the students, whether they comply or argue the point, can spark an interesting debate about personal security and what “social engineering” can mean on a personal level. Ask any students who *did* write their passwords down to change them.

**Specification reference**
3.6.2.2

Specification content

* Define the term malware.
* Describe how malware can be protected against.
* Describe the following forms of malware: computer virus, Trojan, spyware.

Learning outcomes
Be able to describe malware and methods that are suitable for protecting from malware threats.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities

* This topic works well as a discussion, as students will be aware of some of these topics from their own experiences. They may need to be focused somewhat to ensure that they cover all of the topics on the specification.
* A range of useful online videos are available.

Resource

* [Novalabs cyber security protection game](http://www.pbs.org/wgbh/nova/labs/lab/cyber/)
* [Cyber security threats and solutions](https://www.youtube.com/watch?v=fyh05k83js8)

Specification reference
3.6.3

Specification content
Understand and be able to explain the following security measures biometric measures:

* password systems
* CAPTCHA
* using email confirmations
* automatic software updates.

Learning outcomes
Be able to describe a range of security methods.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities
Most of these methods should be demonstrable within the classroom. Students should be encouraged to discuss the advantages and disadvantages of each and identify where and when each might be most appropriate.

#### 3.7 Relational databases and structured query language

Total time for teaching this section: 6 hours

Specification reference
3.7.1

Specification content

* Explain the concept of a database.
* Explain the concept of a relational database.
* Understand the following database concepts:
	+ table
	+ record
	+ field
	+ primary key
	+ foreign key.
* Understand that the use of a relational database facilitates the elimination of data inconsistency and data redundancy.

Learning outcomes

Understand the basic concepts of a SQL database and the advantages of using a relational database.

Suggested timing (lessons)

1 hour

Possible teaching and learning activities

* This is a primarily theoretical section. However, the teacher could demonstrate the concepts by logging into an online SQL database.
* Data consistency/redundancy can be modelled on a student database and having a student “move to a new house”, having parents marry, and so on.

Resource

* [Notes on databases](https://www.bbc.co.uk/bitesize/guides/zfd2fg8/revision/1) and
* [Notes on relational databases](https://www.bbc.co.uk/bitesize/guides/zvq634j/revision/1)

Specification reference
3.7.2

Specification content

* Be able to use SQL to retrieve data from a relational database, using the commands:
	+ SELECT
	+ FROM
	+ WHERE
	+ ORDER BY..ASC | DESC
* Be able to use SQL to insert data into a relational database using the command:
	+ INSERT INTO table\_name (column1, column2 …) VALUES (value1, value2, …)
* Be able to use SQL to edit and delete data in a database using the commands:
	+ UPDATE table\_name SET column1 = value1, column2 = value2 … WHERE condition
* DELETE FROM table\_name WHERE condition.

Learning outcomes

Be able to use the defined SQL commands to select, insert, edit and delete data.

Suggested timing (lessons)

5 hours

Possible teaching and learning activities

* This should be a practical activity with students given access to a real SQL database. There are online sites available or the teacher may set up a short-term hosted SQL server for the period of teaching this topic. Microsoft Access is unlikely to be suitable for teaching this topic.
* Students should be given the opportunity to run live SQL commands on a variety of sample data. Previous specification examinations and those from AQA AS/A-level papers may form suitable starting points for data sets and problems to explore.

Resource
[Notes on queries and SQL](https://www.bbc.co.uk/bitesize/guides/z37tb9q/revision/1)

#### 3.8 Ethical, legal and environmental impacts including privacy

Total teaching time: 4 hours

This section of the specification is well suited to class discussions, debates with students taking opposing sides of an issue and students completing individual research and perhaps making presentations. Exam questions on this section will be drawn from the following areas:

* cyber security
* mobile technologies
* wireless networking
* cloud storage
* hacking (unauthorised access to a computer system)
* wearable technologies
* computer based implants
* Autonomous vehicles.

Throughout this section, students should be referred back to the need to consider these examples in the context of their ethical, legal and environmental impact on society (not all of these are relevant to each example).

Specification reference
3.8

Specification content
Explain the current ethical, legal and environmental impacts and risks of digital technology on society. Where data privacy issues arise, these should be considered.

Learning outcomes

* Students understand the key identified areas in general principles.
* Students should be aware that citizens generally value their privacy.
* Students should be aware that governments and security services argue for access to private data.

Suggested timing (lessons)

4 hours

Possible teaching and learning activities

* Students should be aware of applications and concepts of the technologies identified.
* Students could research applications and risks and put together presentations either individually or with students contributing towards a group presentation.
* Students should be made aware of ethical, legal and environmental impacts and what these mean.
* Students should have practice writing and feeding back on written arguments for and against each of the topics identified in relation to their ethical, legal and environmental impacts as well as their privacy considerations.

Resource

* There are many videos on hacking on YouTube, for example:
* [5 most dangerous hackers of all time](https://www.youtube.com/watch?v=7UaPL5PGywo)
* [10 biggest computer hacks of all time](https://www.youtube.com/watch?v=oOoMqgnvZaY)
* [Resources from the UK government](http://www.gov.uk/government/uploads/system/uploads/attachment_data/file/410221/bis-15-77-Guide-to-cyber-security-schools-programmes-and-resources.pdf)
* [Article on risks of wireless networks](http://ccm.net/contents/805-risks-related-to-wireless-wifi-networks-802-11-or-wi-fi)
* [Video on mobile technology](https://www.youtube.com/watch?v=cl7ccx8oBfw)
* [Examples of implants](https://www.makeuseof.com/tag/plugging-brain-body-future-implanted-computers/)
* [Downloading into brains (video)](https://www.youtube.com/watch?v=P674CG9mOTs)
* [TED talks on wearable technologies](https://www.youtube.com/watch?v=u8tnYt30L-A)

#### Assessment and exam preparation

Total teaching time: 10 hours

It’s important that students are formally assessed on both their programming skills and their theoretical knowledge.

Completing programming tasks under timed conditions and without teacher assistance will help the teacher to identify students who are finding the work challenging so that they can intervene early to help.

While this is an updated specification, much of the content from the previous specification 8520, can be used either directly or with some modification.

1. Assessments are available in either C#, Python or VB.Net [↑](#footnote-ref-1)
2. Content from 3.3.1 and 3.3.2 taught synoptically [↑](#footnote-ref-2)